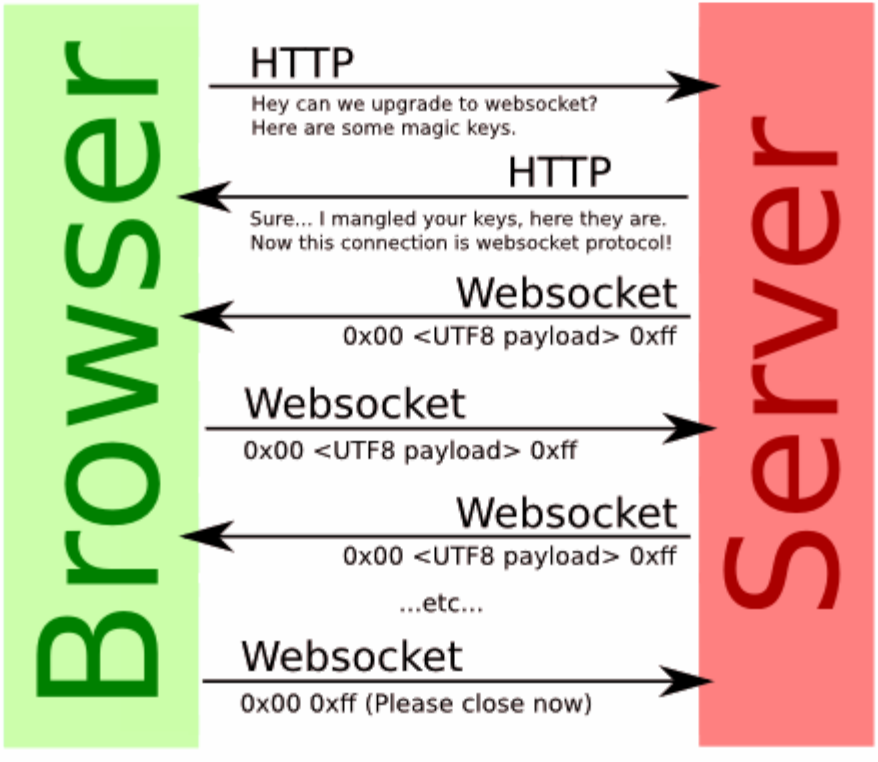
Protocoles web (et +)

HTTP & Websocket



M. PERREIRA DA SILVA

Version PDF des slides

HTTP ?

- **H**yper**T**ext **T**ransfert **P**rotocole
- **Objectif**: accéder à et manipuler des ressources (via des URL)
- Fonctionne par dessus TCP/IP
- Basé lignes de **texte**
  - On peut communiquer avec un serveur HTTP via `telnet` sur le port 80
- **Sans état**
  - N'a aucune mémoire des transactions passées
- Fonctionnement en mode client / serveur
  - **Requête** / **réponse**
- Existe en version sécurisée (HTTPS)

Historique (1/2)

- **1990**: HTTP 0.9
  - **Seulement la méthode GET** (récupération d'une ressource)
  - Pas de code de retour ou de type de fichiers : ne sait renvoyer que du `text/plain`
- **1996**: HTTP 1.0 (RFC 1945)
  - **Nouvelles métodes** de communication (POST, PUT, etc.)
  - Entêtes, types MIME, authentification, etc.
- **1997**: HTTP 1.1 (RFC 2068)
  - Modernisation: **connexions persistantes**, meilleur gestion du cache, requètes partielles, compression ,etc.

Historique (2/2)

- **1999**: Maj de HTTP 1.1 (RFC 2616)
  - **Négociation de contenu**
- **2011**: WebSocket (RFC 6455)
  - **Communication full-duplex**
  - Envoi de données en mode "Push"
  - Passage des serveurs proxy facilité
- **2014**: **Clarification** de HTTP 1.1 (RFCs 7230-7237)
- **2015**: HTTP 2.0 (RFC 7540)
  - Objectif principal: **accélérer le web** (dérivé de google SPDY)

Pourquoi étudier HTTP ?

- C'est la **base de la communication** entre le navigateur et le serveur
- **Nécessaire** coté client lors par exemple de l'envoi de requêtes AJAX
- **Indispensable** coté serveur pour
  - Récupérer les données envoyées par le client
  - Gérer les sessions
  - Gérer la mise en cache des informations
  - etc.

Exemple de requête HTTP

```
GET / HTTP/1.1
Host: www.perdu.com
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: fr;q=0.8,en;q=0.6
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/41.0.2272.101 Safari/537.36

```

Attention: la requête termine par une ligne vide (en jaune)

Exemple de réponse HTTP

```
HTTP/1.1 200 OK
Date: Thu, 26 Mar 2015 14:48:50 GMT
Server: Apache
Last-Modified: Tue, 02 Mar 2010 18:52:21 GMT
Etag: "cc-480d5dd98a340"
Content-Type: text/html
Content-Length: 204
Accept-Ranges: bytes

<html><head><title>Vous Etes Perdu ?</title></head><body><h1>Perdu sur l'Internet
?</h1><h2>Pas de panique, on va vous aider</h2><strong><pre>    * <----- vous
&ecirc;tes ici</pre></strong></body></html>
```

Attention: une ligne vide (en jaune) sépare les entêtes des données renvoyées par le serveur

Structure d'une requête

1. **Ligne de requête**: `METHOD URL [HTTP-version]`*
   - Ex: `GET /index.html HTTP/1.1`
2. **Entêtes** (ordre conseillé mais par obligatoire)
   - Entêtes **généraux**: applicables aux requêtes **et** réponses
     - Ex: `Date: Thu, 26 Mar 2015 14:48:50 GMT`)
   - Entêtes de **requête**: spécifiques aux requêtes
     - Ex: `Accept: text/html`
   - Entêtes d'**entité**: méta-informations concernant le corps du message
     - Ex: `Content-Length: 204`
3. **Ligne vide**
4. Corps du message (**données**)
   - Ex: texte, code html, données de formulaire, image, etc.

red[*] Par défaut c'est la version 1.0 de HTTP qui est utiisée

Les principales méthodes HTTP

- `OPTIONS` : demande les **options de communication** disponibles
- `GET` : demande des **informations** (entêtes) et des **données** (corps de la réponse) concernant la ressource située à l'URL spécifiée
  - Utilisé lorsque vous consultez une page web
- `HEAD` : demande des **informations** (entêtes) concernant la ressource située à l'URL spécifiée
- `POST` : **envoie des donnée**s (contenu d'un formulaire) qui seront traitées par le script / programme situé à l'URL
  - Utilisé lorsque vous envoyez les données d'un formulaire (généralement)
- `PUT` : **stocke** des données à l'URL spécifiée
  - Utilisé lorsque vous envoyez un fichier vers un serveur
- `DELETE` : **supprime** les données situées l'URL spécifiée
- `TRACE` : demande au serveur de retourner au client les données qui lui ont été envoyées (**écho**)

Rappel sur les URL

- **Absolue** : `schéma://utilisateur:motdepasse@domaine:port/chemin?requête#fragment`
  - Ex: `http://joe:bar@www.univ-nantes.fr:80/polytech/dpts/info?enseignant=perreiradasilva-m&cours=technos-web#slide4`
- **Relative** : `chemin?requête#fragment`
  - Ex1: `ici`
  - Ex2: `/ici`
  - Ex3: `./ici?query=something`
  - Ex4: `../la/fichier.html`
- Ne sont codées qu'à partir d'un **jeu limité** de caractères. Pour le reste, on utilise le **percent encoding**

Les types MIME (`Content-type`)

- **M**ultipurpose **I**nternet **M**ail **E**xtensions (rfc 2045-2046)
  - Extension du format des emails pour supporter autre chose que le texte ASCII
  - Définit le format du contenu du mail et de ses pièces jointes
- Type MIME : partie `Content-type` de MIME
  - Syntaxe : `type/sous-type`
  - Exemples :
    - Texte: `text/plain`, `text/html`
    - Fichiers pluri-usages: `application/pdf`, `application/xml`
    - Multimédia: `audio/mpeg`, `image/jpeg`, `video/mp4`
    - Etc.

Requête : les entêtes spécifiques 1/3

- `Accept` : types MIME acceptés
  - Ex: `Accept: text/plain; q=0.5, text/html; q=0.8`
- `Accept-Charset` : encodages de caractères acceptés
  - Ex: `Accept-Charset: iso-8859-5, unicode-1-1;q=0.8`
- `Accept-Encoding` : types de compression (éventuelle) acceptés
  - Ex: `Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0`
- `Accept-Language` : langues acceptées
  - Ex: `Accept-Language: fr; q=1.0, en; q=0.5`

Pour tous les entêtes de type `Accept`, on peut spécifier une préférence `q` *($q \in [0,1]$)* qui sera utile lors de la négociation de contenu.

- `Authorization` : informations d'authentification auprès d'un serveur web
  - Ex: `Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`
- `Expect` : demande d'un comportement particulier du serveur
  - Ex: `Expect : 100-continue`

Requête : les entêtes spécifiques 2/3

- `From` : adresse mail de l'utilisateur du client
  - Ex: `From: matthieu.perreiradasilva@univ-nantes.fr`
- `Host` : adresse (et port) de l'hote à qui est destiné la requête
  - Ex: `Host: web.polytech.univ-nantes.fr:80`
- `If-Match` : rend la requête conditionnelle à un tag d'entité `ETag`
  - Ex: `If-Match : "686897696a7c876b7e"`
- `If-Modified-Since` : rend la requête conditionnelle à une date de modification
  - Ex: `If-Modified-Since: Fri, 03 Apr 2015 12:00:00 GMT+2`
- `If-None-Match` : inverse de `If-Match`
- `If-Range` : requête conditionnelle (date ou Etag) sur une partie d'une ressource (cf. header `Range`)
- `If-Unmodified-Since` : Inverse de `If-Modified-Since`

Requête : les entêtes spécifiques 3/3

- `Max-Forward` : nombre maximal de fois qu'un message peut être transféré (cf. méthode `TRACE`)
  - Ex: `Max-Forwards : 5`
- `Proxy-Authorization` : informations d'authentification auprès d'un proxy
  - Ex: `Proxy-Authorization: Basic dXNlbWU6dGVzdA==`
- `Range` : ne demande qu'une partie d'une ressource
  - Ex: `Range: bytes=500-999`
- `Referer` : adresse à partir de laquelle la requête provient
  - Ex: `Referer: http://www.univ-nantes.fr/index.html`
- `TE` : indique au serveur quel type de transfert de données par bloc le client supporte
  - Ex: `TE: deflate`
- `User-Agent` : informations identifiant l'agent utilisateur (navigateur) utilisé.
  - Ex: `Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Win64; x64; Trident/6.0)`

Structure d'une réponse

1. **Ligne de réponse**: `HTTP-version Status-code Reason-phrase`
   - Ex: `HTTP/1.1 200 OK`
2. **Entêtes** (ordre conseillé mais par obligatoire)
   - Entêtes **généraux**: Applicables aux requêtes **et** réponses
     - Ex: `Date: Thu, 26 Mar 2015 14:48:50 GMT`)
   - Entêtes de **réponse**: Spécifiques aux réponses
     - Ex: `Server: Apache`
   - Entêtes d'**entité**: méta-informations concernant le corps du message
     - Ex: `Content-Length: 204`
3. **Ligne vide**
4. Corps du message (**données**)
   - Ex: texte, code html, données de formulaire, image, etc.

Ligne de statut

- `Status-code` : code numérique représentant l'état de la requête (succès, échec, etc.)
  - 1xx: Informations
  - 2xx: Succès
  - 3xx: Redirection
  - 4xx: Erreur client
  - 5xx: Erreur serveur
- `Reason-phrase` : texte expliquant le `Status-code`

Les principaux codes de statut

Code	Signification	Code	Signifiation	Code	Signifiation
100	Continue	101	Switching protocols		
200	**OK**	201	Created	202	Accepted
203	Non-Authoritative Information	204	No Content	205	Reset Content
206	Partial Content				
300	**Multiple Choices**	301	Moved Permanently	302	**Found**
303	See Other	304	**Not Modified**	305	Use Proxy
306		307	**Temporary Redirect**	308	Permanent Redirect
400	**Bad Request**	401	**Unauthorized**	402	Payment Required
403	**Forbidden**	404	**Not Found**	405	Method Not Allowed
406	Not Acceptable	407	Proxy Authentication Required	408	Request Timeout
409	Conflict	410	**Gone**	411	Length Required
412	Precondition Failed	413	Request Entity Too Large	414	Request-URI Too Long
415	Unsupported Media Type	416	Requested Range Not Satisfiable	417	Expectation Failed
500	**Internal Server Error**	501	**Not Implemented**	502	Bad Gateway
503	**Service Unavailable**	504	Gateway Timeout	505	HTTP Version Not Supported

Réponse: les entêtes spécifiques 1/2

- `Accept-Ranges` : indique si le serveur accepte les envois partiels
  - Ex: `Accept-Ranges: bytes` ou `Accept-Ranges: none`
- `Age` : age (en secondes) des données renvoyées (utile si les données sont en cache)
  - Ex: `Age: 1030`
- `ETag` : chaine (hash) identifiant de manière unique **une version** d'une ressource.
  - Ex: `ETag: "686897696a7c876b7e"`
- `Location` : demande au client d'effectuer une redirection vers l'URL fournie
  - Ex: Location: `http://www.nouvelle-adresse.fr`
- `Proxy-Authenticate` : Demande d'authentifiation du client par un proxy
  - Ex: `Proxy-Authenticate: Basic realm="WallyWorld"`
- `Retry-After` : demande de re-tenter la requête après n secondes (après erreur 503: Service unavailable)
  - Ex: `Retry-After: 120`

Réponse: les entêtes spécifiques 2/2

- `Server` : information concernant le serveur qui a répondu à la requête
  - Ex: `Server: Apache/2.2.14 (Win32)`
- `Set-Cookie` : ensemble de paires clé/valeur à stocker sur le client + options
  - Ex: `Set-Cookie: name1=value1,name2=value2; Expires=Wed, 09 Jun 2021 10:18:14 GMT`
- `Vary` : informe le client qu'il existe plusieurs "versions" de la ressource demandée (cf. négociation de contenu)
  - Ex: `Vary: Accept-Language, Accept-Encoding`
- `WWW-Authenticate` : Demande d'authentifiation du client par un serveur web
  - Ex: `WWW-Authenticate: Basic realm="WallyWorld"`

Les entêtes généraux 1/2

- `Cache-Control` : permet de spécifier une politique de cache que **doivent** respecter clients ou serveurs
  - Ex: `Cache-control: no-cache`
- `Connection` : contrôle de la connection HTTP (fermer, garder ouverte, etc.)
  - Ex: `Connection: close`
- `Date` : date d'envoi du message
  - Ex: `Date: Thu, 26 Mar 2015 14:48:50 GMT`
- `Pragma` : directive spécifique, plus utilisée
- `Trailer` : déclare qu'un certain nombre de champs seront présents à la fin du message plutôit que dans le header (transfert par blocs seulement)
  - Ex: `Trailer: Max-Forwards`

Les entêtes généraux 2/2

- `Transfer-Encoding` : méthode d'encodage des données
  - Valeurs autorisées: `chunked`, `compress`, `deflate`, `gzip` ou `identity`
- `Upgrade` : demande un changement de protocole
  - Ex: `Upgrade: websocket`
- `Via` : utilisé par les passerelles et proxy pour spécifier des protocoles intermédiaires
- `Warning` : informations complémentaires concernant le message
  - Ex: `Warning: 112 Disconnected Operation`

Les entêtes d'entité

- `Allow` : liste des méthode supportée pour accéder à la ressource
  - Ex: `Allow: GET, HEAD, PUT`
- `Content-Encoding` : type d'encodage utilisée pour encoder l'entité (ressource)
  - Ex: `Content-Encoding: gzip`
- `Content-Language` : langue utilisée pour représenter l'entité
  - Ex: `Content-Language: fr`
- `Content-Length` : taille de la ressource (en octets)
  - Ex: `Content-Length: 6894`
- `Content-Location` : URL à partir de laquelle on peut accéder à la ressource (si différente de l'URL de la requète)
  - Ex: `Content-Location: http://www.univ-nantes.fr/content.json`

* entité = données transmises dans le corps de la requête ou la réponse

Les entêtes d'entité 2/2

- `Content-MD5` : hash MD5 de la ressource, pour vérification éventuelle de son intégrité
  - Ex: `Content-MD5  : 8c2d46911f3f5a326455f0ed7a8ed3b3`
- `Content-Range` : En cas d'envoi partiel de la ressource, indique la plage de données envoyée.
  - Ex: `Content-Range : bytes 500-999/1234`
- `Content-Type` : Type MIME de la ressource envoyée, suivi optionellement de paramètres
  - Ex: `Content-Type: text/html; charset=utf-8`
- `Expires` : indique la date/heure à partir de laquelle la réponse est consisérée comme obsolète
  - Ex: `Expires: Wed, 01 Apr 2015 23:59:59 GMT+2`
- `Last-Modified` : date/heure à laquelle la ressource a été modifiée pour la dernière fois
  - Ex: `Last-Modified: Tue, 31 Mar 2015 12:12:12 GMT+2`

Retour sur la négociation de contenu

- Possibilité d'avoir plusieurs versions d'une même ressource
  - On peut demander au serveur des informations sur la ressource via une requête `HEAD`. Le header `Vary` indique sur quels critères on peut négocier.
  - Le client spécifie ses préférences avec les entêtes `Accept-*` lors de sa requête `GET`
  - Si le serveur possède une version adéquate il la renverra. Sinon il renvoie un code d'erreur `406 Not Acceptable`
  - On peut "négocier": la langue, l'encodage des caractères, le type MIME, la compression
- **Astuce** pour les fichiers **image, audio et video**: si on ne précise pas d'extension dans son fichier HTML, le navigateur va négocier le format le plus approprié.

Les cookies 1/2

- HTTP est un protocole sans état
  - Chaque requête est indépendante de la précédente
  - Comment garder une trace des transactions passées (gestion de **session**, **personnalisation**, **tracking**) ?
- Les cookies HTTP !
  - Entête `Set-Cookie` envoyé par le serveur pour stocker des informations sur le client
  - Entête `Cookie` utilisé par le client pour renvoyer vers le serveur les informations stockées précédement
  - Taille d'un cookie limitée à 4096 octets
  - Max 50 cookies par domaine (site web)
  - Les cookies ont une date d'expiration

Les cookies 2/2

- Sécurité...
  - Un serveur ne peut récupérer que les cookies qu'il a lui même demandé au client de stocker (même domaine)
- Alternatives
  - Passage d'identifiant de session dans la partie *query string* l'URL (`GET`)
  - Passage d'identifiant de session dans un champ de formulaire caché (`POST`).
  - Local storage et session storage de HTML5

Les cookie: exemple

Requête du client:

```
GET /index.html HTTP/1.1
Host: www.univ-nantes.fr
...
```

Réponse du serveur:

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=blue
Set-Cookie: sessionToken=fds45fds2; Expires=Fri, 26 Jun 2017 10:10:30 GMT+2
...
```

Autre requête (ultérieure) du client:

```
GET /cours.html HTTP/1.1
Host: www.univ-nantes.fr
Cookie: theme=blue; sessionToken=fds45fds2
...
```

Les webSockets

- Pourquoi ?
  - HTTP est basé sur une architecture client / serveur
    - Le serveur ne peut **que répondre** à une requête du client
  - Pas de vraies possibilités communication bi-directionnelle (ex: *Push*)
    - *Polling* et *long-polling* = requêtes périodiques de l'état d'une ressource
- C'est quoi ?
  - Protocol de communication faisant partie des spécifications HTML5
  - Complémentaire à HTTP
    - **Canal de communication bi-directionnelle** (*full-duplex*)
    - Moins verbeux
    - Latence moindre
    - Traverse simplement les proxy et firewalls

Le protocole

- Doit fonctionner avec l'infrastructure web existante
  - **Démarre par une requête HTTP**
  - Demande de **changement de protocole** via l'entête `Upgrade` et un échange de clés `Sec-WebSocket-Key` et `Sec-WebSocket-Accept`
  - Passage ensuite en websocket sur la **même connexion TCP/IP** (par défaut port 80 ou 443)
- Contrairement à HTTP, les trames WebSocket utilisent un **encodage binaire**
- Très **peu de données additionnelles liées au protocole** (entête, etc.) : quelques octets

* Le support de WebSocket dans les navigateurs est assez récent. Attention aux éventuels problèmes de compatibilité

WebSockets et JavaScript

- Coté **navigateur**
  - Fait partie de **HTML5**
  - **API** clairement définie et universelle
- Coté **serveur web**: dépendant de la technologies utilisée
  - Apache WebSocket module: module (écrit en C) pour apache
  - pywebsocket4 : implémentation en python, pour apache également
  - jWebSocket
  - Socket.io, **ws**: modules WebSockets pour **nodejs**
  - Etc.

Websockets coté navigateur (HTML5)

Exemple minimaliste

```
// ouverture de la connexion
var ws = new WebSocket("ws://exemple.polytech-nantes.fr");

// on déclare un "callback" qui sera appelé à l'ouverture de la connexion
ws.on('open', function() {
    // envoi d'un message de bienvenue
    ws.send('Je suis un navigateur !');
});

// autre "callback" appelé à la réception d'un message
ws.on('message', function(message) {
    console.log('Message reçu (client): %s', message);
});
```

Websockets coté serveur web (ws/nodejs)

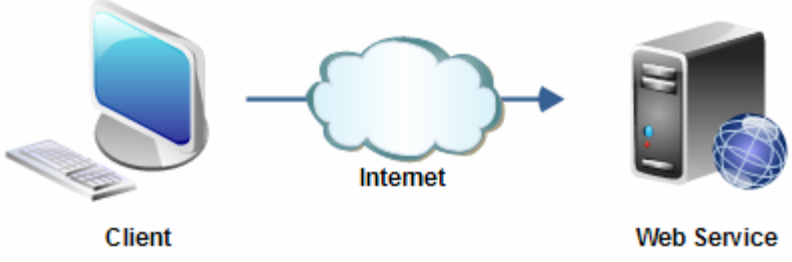
Exemple minimaliste

```
// import du module nodejs "ws"
var WebSocketServer = require('ws').Server

// ouverture de la connexion (ici sur le port 8080)
var wss = new WebSocketServer({port: 8080});

// callback appelé à l'ouverture de la connexion
wss.on('connection', function(ws) {
    // un fois la connexion ouverte, on déclare un  "callback" qui sera appelé à
la réception d'un message
    ws.on('message', function(message) {
        console.log('Message reçu (serveur): %s', message);
    });
    // on envoie ensuite un message au navigateur qui vient de se connecter
    ws.send('Moi, je suis un serveur...');
});
```

Services web

C'est quoi ?

- Un service Web est une **application logicielle** accessible à partir du **web**. Il utilise les **protocoles internet** pour communiquer et utilise un **langage standard** pour décrire son interface



- Quelle différence avec un site web ?
  - La **présentation** des informations est inutile (HTML)

La définition du W3C

> *Un service web est un **système logiciel** identifié par **un URI**, dont les interfaces publiques et les « bindings » sont définies et décrites en **XML**. Sa définition peut être **découverte** [dynamiquement] par d'autres **systèmes logiciels**. Ces autres systèmes peuvent ensuite interagir avec le service web d'une façon décrite par sa définition, en utilisant des **messages XML** transportés par des *protocoles Internet.*

* Le W3C met en avant XML comme langage de description (à différents niveaux). Mais ce n'est pas le seul moyen (standard) de communication...

Exemples...

- Nombreuses **API** disponibles sur le web
  - Google: https://developers.google.com/apis-explorer (REST)
  - Twitter: https://dev.twitter.com/ (REST)
  - Facebook: https://developers.facebook.com/ (REST)
  - Paypal: https://developer.paypal.com (SOAP)
  - Viamichelin: http://dev.viamichelin.fr/presentation-soap.html (SOAP)
- Accès à ces API via les protocoles de service web (SOAP, REST, etc.)

ou

- Via des kit de développement (SDK) spécifiques (JavaScript, Java, etc.)
  - Les SDK ne font que simplifier l'accès au service web

Les formats d'échange de données web

XML

- **Extensible Markup Language**
- Vu dans les cours précédents (*Remi Lehn*)
- Langage de **description** de données
- On peut **créer de nouveaux langages** via un Schéma XML
- On peut **transformer** un document XML vers d'autres formats (XML ou non) via XSLT
- La base de nombreux services web
- **Attention:** format particulièrement *verbeux*

JSON

- Format de description de données textuel
  - Existe en version binaire (BSON)
- Dérivé de la **notation objet de JavaScript**
- Ne contient que 2 types de structures
  - **Objet** = ensemble non ordonnée de paires `"clé" : valeur`
  - Ex: `{ "nom": "Polytech", "nbEtudiants: 500" }`
  - **Tableau** = collection ordonnée de valeurs
  - Ex: `[ 1, 2, 3, 4, 5]`
- Une **valeur** peut être:
  - Un objet
  - Un tableau
  - Un type simple (chaine, nombre, booléen, null)

XML vs. JSON: exemple

XML

```xml
<product>
     <id>15</id>
     <name>Widgets</name>
     <description>These widgets are
the finest widgets ever made by
anyone.</description>
     <options type="color">
          <item>Purple</item>
          <item>Green</item>
          <item>Orange</item>
     </options>
</product>
```

JSON

```json
"product" : {
     "id" : 15,
     "name" : "Widgets",
     "description" : "These widgets
are the finest widgets ever made by
anyone.",
     "options" : [
          {
               "type" : "color",
               "items" : [
                    "Purple",
                    "Green",
                    "Orange"
               ]
          }
     ]
}
```

REST ? (1/2)

- REST: **REpresentational State Transfer**
- **Style d'architecture** != protocole de communication (SOAP)
- Décrit en 2000 par Roy Thomas Fielding dans sa thèse, chap 5, "Architectural Styles and the Design of Network-based Software Architectures"
  - 1 des principaux acteur de la spécification de HTTP
  - Membre fondateur de la fondation Apache
  - Développeur du serveur Web Apache
- Style d'architecture **inspiré du Web**
- Architecture **orientée ressource**

REST ? (2/2)

- Utilisé pour développer des **Services Web**
- Dans la littérature
  - Architectures orientées **ressources** (ROA)
  - Architectures orientées **données** (DOA)
- Quand une application respecte ces principes: **RESTFul**
- Les services web REST sont **sans états** (Stateless)
  - Pas de mémoire des requêtes antérieures
  - Chaque requête envoyée doit contenir toutes les informations nécessaires au traitement

Les concepts de REST

- **Ressources** (Identifiant)
  - Identifiée par une URI
  - Exemple (fictif): http://www.achats.fr/livre/SF/Harry-Potter
- **Méthodes** (Verbes)
  - Action à effectuer sur la ressource
  - Méthodes HTTP: `GET`, `POST`, `PUT` et `DELETE`
- **Représentation** (Vue de la ressource ou de son état)
  - Informations échangées avec le service
  - Texte, XML, JSON, ...

REST: Ressource

- Tout ce qui est **identifiable / manipulable** dans le système
  - Document, Image, Personne, Le montant du compte d'un client, etc.
- **Identifié par un lien** (URI)
- Une ressource **peut avoir plusieurs URI**
- Une URI identifie **une seule ressource** (ou un seul groupe de ressources)
- Construite de façon **hiérarchique**
- La représentation d'une ressource **peut évoluer avec le temps**
  - Lié au temps: ex. Dernier article
  - Modification structure: ex. Ajout d'un champ

REST: Ressource

- Structure classique
  - Structure **hiérarchique**
  - Construction classique
  - `http://domaine.com/<plus général>/../<plus spécifique>`
- Exemples d'URIs
  - /musique/rock
  - /musique/rock/AC-DC/
  - /musique/rock/AC-DC/année
  - /musique/rock/AC-DC/back_in_black
  - /musique/rock/AC-DC/année/5
  - /musique/classique/meilleures_ventes
  - /musique/recherche/foxy_lady

REST: Interface / methodes

- REST fournit une interface uniforme
- Chaque ressource peut subir **4 opérations de base** (CRUD)
  - Create (Créer)
  - Retrieve (Lire)
  - Update (MAJ)
  - Delete (Supprimer)
- REST **s'appuie sur HTTP** pour exprimer les opérations via les méthodes HTTP
  - `POST` (Créer)
  - `GET` (Lire)
  - `PUT` (MAJ)
  - `DELETE` (Supprimer)
- Il n'est pas nécessaire d'implémenter toutes les méthodes pour une ressource

REST: Représentation

- **Format d'échange** des données
  - Pour le client (`GET`)
  - Pour le serveur (`PUT` ou `POST`)
- Généralement: Texte, JSON, XML, HTML, CSV
- Le format d'entrée (`POST`) et de sortie (`GET`) d'une même ressource peut varier
- On peut spécifier le format via
  - **Entêtes HTTP** (type MIME via `content-type`)
  - **L'URL** de la ressource (`/musique/rock/xml` ou `/musique/rock/json`)

REST: Limites

- **Actions binaires**:
  - Demander si un artiste fait partie d'un groupe ?
  - Demander si un groupe contient un artiste ?
  - Plus dur : lien entre groupe et artiste lors de la création ?
- **Transactions**
  - Décrire un virement bancaire ?
  - Créer une ressource incrémentalement ?
- Les URI doivent-elles être **évidentes ou opaques** ?
  - Donnée: `/serveur/home/~maurice/` ("joli", mnémotechnique)
  - Identifiant: `/homedir/23eab89c/` (résistant au changement)
- **Sécurité**
  - Authentification HTTP simple
  - Utilisation d'HTTPS, cookies, tokens, etc.